CS3, Fall 1996
Midterm #2
Professor Grillmeyer

Problem #1 (20 points)

A) Write a function `exaggerate` that takes a list and returns a list with all the top-level numbers in the argument list squared. Here are some sample calls.

```
> (exaggerate '(my car does 0 to 60 mph in 10 seconds))
(my car does 0 to 3600 mph in 100 seconds)
> (exaggerate '(these numbers: 3 4 5 but not these (6 7 8)))
(these numbers: 9 16 25 but not these (6 7 8))
```

```
(define (exaggerate a-list)
```

B) Is your function above tail or embedded recursive?

Problem #2 (14 points)

Someone proposes the following function to work with `exaggerate` that will square numbers deep within sublists as well.

```
(define (exaggerate-all a-list)
   (cond ((null? a-list) '())
         (else (if (list? (first a-list))
                   (cons (exaggerate (first a-list))
                         (exaggerate-all (rest a-list)))
                   (exaggerate-all a-list)))))
```

A) Assuming that exaggerate works properly with numbers on the top-level, show a sample call to exaggerate-all with a two element list that returns a list with **all the numbers squared.** The two blanks below represent the two elements in the argument to exaggerate-all . Numbers should appear somewhere in your answer.

```
(exaggerate-all '( ____________ _____________ ))
```

B) Fill in the blanks to show an example call to exaggerate-all with a two element list that **does not square any of the numbers** in its argument list. Numbers should appear somewhere in your answer.

```
(exaggerate-all '( ____________ ______________))
```

C) Fill in the blanks to show a call to exaggerate-all with a two element list that **produces an error** . Numbers should appear somewhere in your answer. Indicate what the error is as well.

```
(exaggerate-all '( ____________ ______________))

Error: _______________________________
```

Problem #3 (16 points)

Complete the function `num-list` below to return a list of numbers that occur anywhere within a list. For example,

```
> (num-list '(the (1 answer is (always 42))))
(1 42)
```

Complete the function below.

```
(define (num-list a-list)

   (cond ((null? ___________) ____________)

         ((number? ___________) ___________)

         ((atom? ___________) ___________)

         (else
              (___________

              (num-list ___________)

              (num-list ___________)))) )
```

Problem #4 (10 points)

What do the following Scheme expressions evaluate to? If they produce errors, indicate what the errors are. Assume that the following expressions have been entered previously.

```
       (define formula (* 4 (+ 2 3)))
       (define answer '(/ 91 7))
```

Write your answers in the space below each Scheme expression.

```
(atom? formula)                                         (and 0 1 2)
```

```
(number? answer)
```

```
(intersection 421 21)
```

```
(if 0 1 2)
```

```
(cond (0 1 2))
```

```
(or 0 1 2)
```

```
(equal? 1 (rest '(0 1)))
```

```
(cond ((not formula))
      (formula))
```

```
(min '(1 -2 3))
```

Problem #5 (3 points)

Write out the list representation of the tree below usin the format presented in Chapter 7 - leaves are atoms.

```
  1
 / \
2   3
   / \
  4   5
 / \
6   7
```

Problem #6 (12 points)

The function `fringe` takes a binary tree, *tree*, that is in the form presented in Chapter 7 of the reader - lears are atoms. The function returns a list of all the leaves in `tree` . For example,

```
> (fringe '(* (+ 18 67) xyzzy))
(18 67 xyzzy)

> (fringe '())
()
```

`Complete the function fringe`

```
(define (fringe tree)

   (cond ((null? ___________)

                 ___________)

         ((atom? ___________)

                 ___________)

         (else

             (___________

                 (fringe ___________)

                 (fringe ___________)))) )
```

Problem #7 (6 points)

Given the functions below:

```
    (define (abc xyz)                          (define (def uvw)
      (cond ((first xyz) (rest xyz))             (or (zero? (first uvw)) (def (rest
uvw))))
            (else (abc xyz))))
```

A) What does the call `(abc '(#f #t))` return?

B) What does the call `(def '(1 0 2)` return?

Problem #8 (14 points)

Given the two functions below:

```
(define (unknown n)
  (cond ((= n 0) 'stop)
        (else (what n n)
              (unknown (- n 1)))))
```

```
(define (what x y)
  (cond ((= x 0) (newline) x)
        (else (display y)
              (what (- x 1) y))))
```

A) Show all output from **display** and **newline** when the call `(unknown 4)` is made? **Do not show the final return value.**

B) What is the **return value** of the call `(unknown 100)` ?

C) Write out the output from **display** and **newline** from the call `(unknown 4)` given that the actions in the `else` clause of `unknown` are reversed to be

```
       (else (unknown (- n 1))
             (what n n))
```

D) What is the **return value** of the above call to the new `unknown` ?

E) Now we'll swap the else actions of `what` so it is

```
       (else (what (- x 1) y)
             (display y))
```

Show the **output and the return value** as the computer would print of the call `(what 3 3)` .

Posted by HKN (Electrical Engineering and Computer Science Honor Society)
University of California at Berkeley
If you have any questions about these online exams
please contact mailto:examfile@hkn.eecs.berkeley.edu